Transformer & Credibility

Lecture 12

36th International Summer School SAA

University of Lausanne

Ronald Richman, Salvatore Scognamiglio, Mario V. Wüthrich

12 September 2025

Attention is All You Need

- The most important paper in machine learning history?
- Introduced the Transformer architecture
- Currently powering AI applications:
  - GPT5 = Generative Pretraining Transformer
  - Claude
  - Gemini
  - Grok
- Some questions:
  - Can we understand Transformer models in an actuarial manner?
  - Can these models be applied for actuarial work?
  - Can we improve them with actuarial techniques?

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* † illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Roadmap of Talk

- Introduce Transformer models
- Explain how to read the paper "Attention is All You Need"
- Discuss how to adapt these to tabular data $=$ actuarial problems
- Discuss credibility and show how the Transformer can be reworked as a credibility model
- Show how the model can be interpreted and improved

European Actuarial Journal
https://doi.org/10.1007/s13385-025-00413-y

ORIGINAL RESEARCH PAPER

Check for updates

The credibility transformer

Ronald Richman[1] · Salvatore Scognamiglio[2] · Mario V. Wüthrich[3]

Received: 5 October 2024 / Revised: 28 December 2024 / Accepted: 20 January 2025

Abstract
Inspired by the large success of Transformers in Large Language Models, these architectures are increasingly applied to tabular data. This is achieved by embedding tabular data into low-dimensional Euclidean spaces resulting in similar structures as time-series data. We introduce a novel credibility mechanism to this Transformer architecture. This credibility mechanism is based on a special token that should be seen as an encoder that consists of a credibility weighted average of prior information and observation based information. We demonstrate that this novel credibility mechanism is very beneficial to stabilize training, and our Credibility Transformer leads to predictive models that are superior to state-of-the-art deep learning models.

Keywords Transformer · Credibility · Tabular data · Feature-engineering · Entity embedding

1 Introduction to Transformers

2 Understanding Attention is All You Need

3 Seq2Seq to Tabular Data

4 Credibility

5 Example + Interpretability

6 Improvements

7 Conclusions

Transformer

- General purpose model for processing data for machine learning
- Data = text, images, sound or tabular data = data used for actuarial work
- Relies on several main components covered earlier:
  - Embedding data into vector space
  - Self-attention operation...
  - ... and extension to multi-head attention
  - Feedforward networks
  - Skip connections
  - Normalization
- We will provide intuitive explanations for all of these in context of non-life pricing example

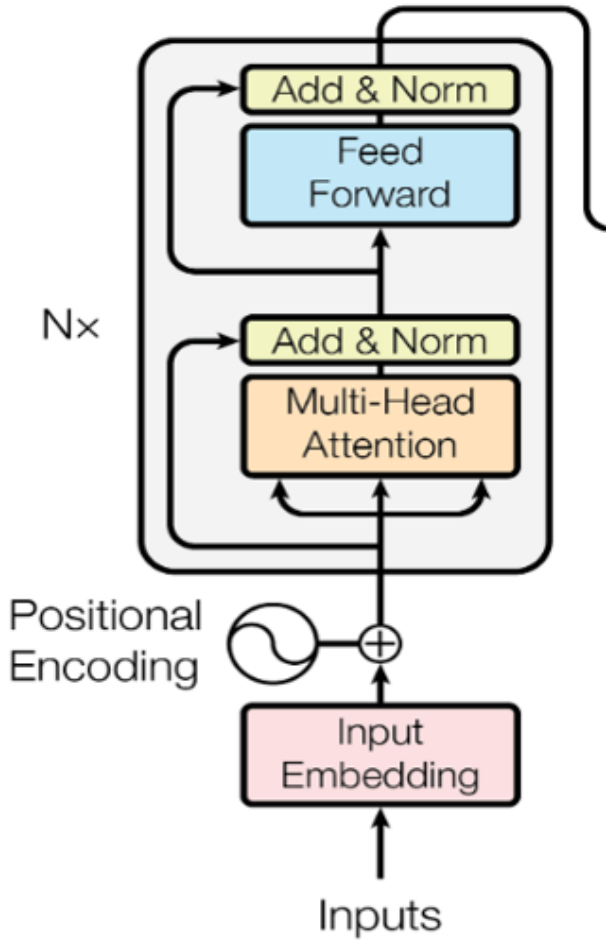
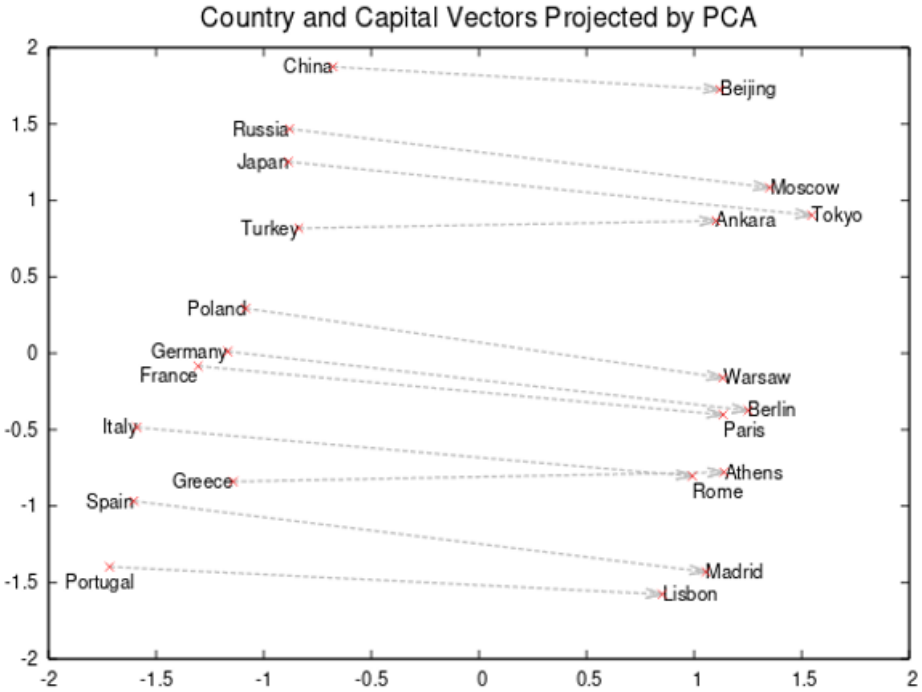


Image from "Attention is all you need"

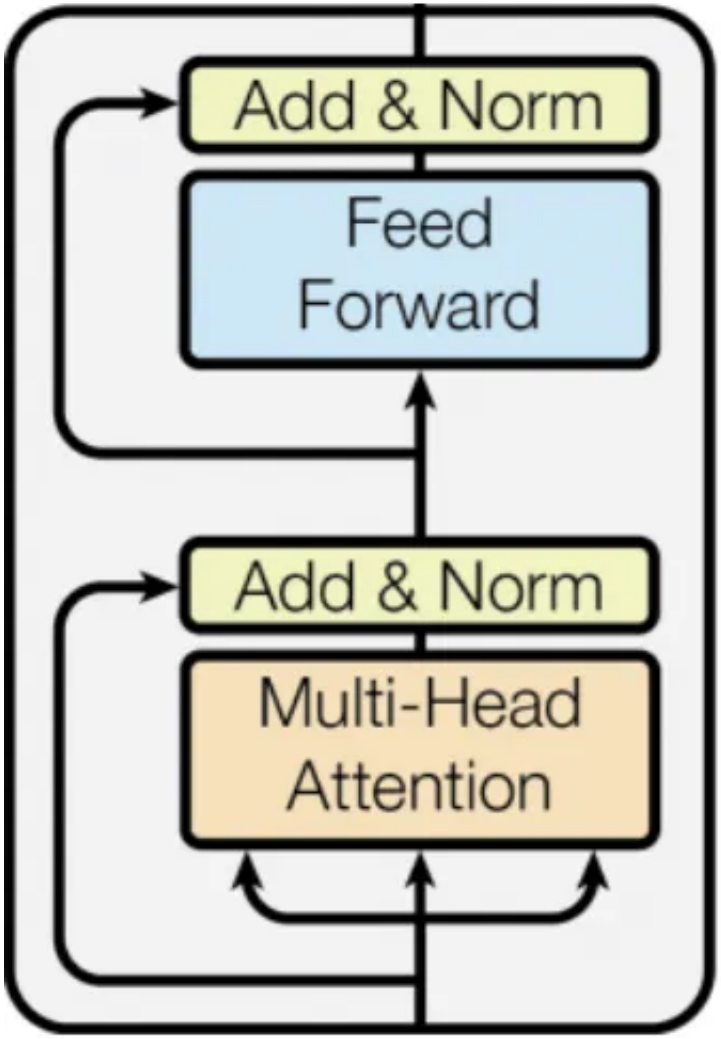
Modern ML relies on embeddings!

- Transformer was a new approach for deep learning relying on learning embeddings for NLP and applying Transformers to these
- This approach was proposed in 2017 and relies on attention mechanisms
- Extended to other NLP tasks, computer vision and more recently, tabular data



Transformers = Compute over Embeddings

- Inputs are mapped to embeddings (tokens, image patches, tabular features) (Vaswani et al., 2017)
- Self-attention performs content-dependent routing/mixing between embeddings via learned $\boldsymbol{Q}/\boldsymbol{K}/\boldsymbol{V}$ projections (Vaswani et al., 2017)
- Feed-forward layers apply non-linear per-token compute; stacking layers composes these computations (Geva et al., 2021)
- Positional/feature encodings make the compute structure-aware (order or field identity) (Vaswani et al., 2017)



Encoder layer

1 Introduction to Transformers

2 Understanding Attention is All You Need

3 Seq2Seq to Tabular Data

4 Credibility

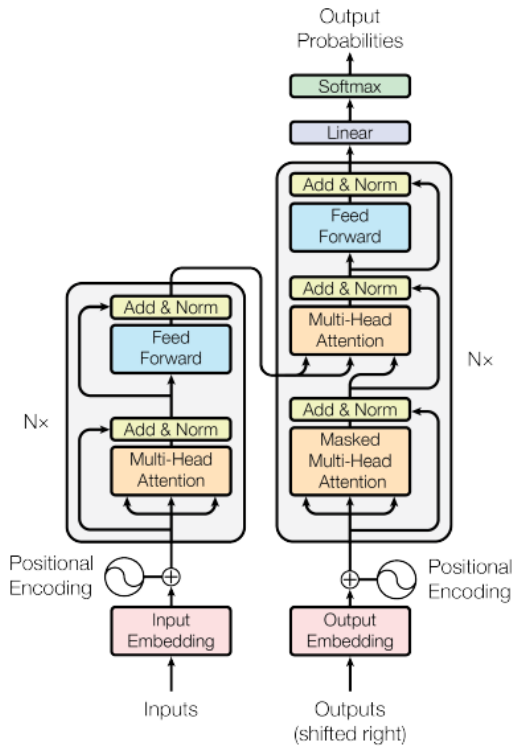
5 Example + Interpretability

6 Improvements

7 Conclusions

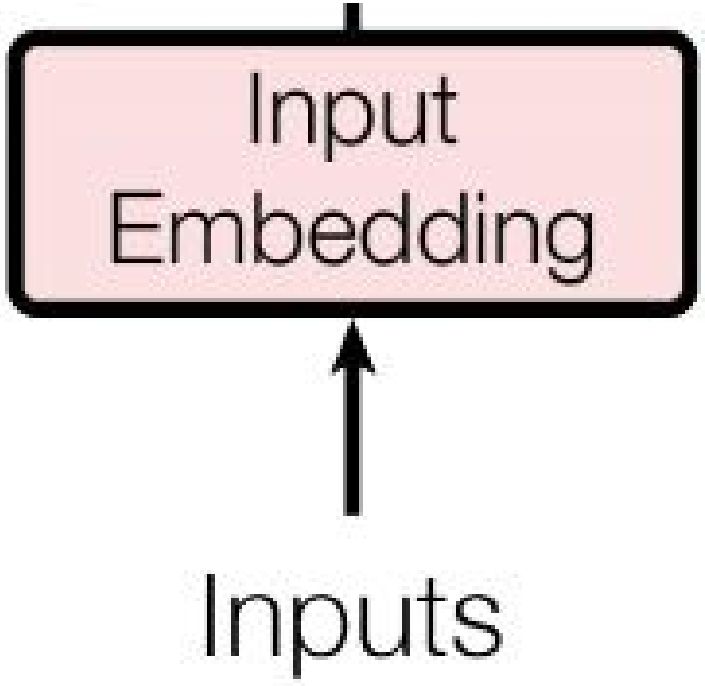
Transformer Overview: Attention is All You Need

- The Transformer architecture from "Attention is All You Need" focuses on sequence-to-sequence modeling,
- such as translating an English sentence to French.
- It accomplishes this by using the earlier implementation of self-attention in Bahdanau, Cho, and Bengio (2015).
- Key concepts include:
  - **Inputs + Embeddings**
  - **Encoder**
  - **Decoder**
- Each of these blocks are composed of several components i.e. this is a compound architecture!



Input and Input Embeddings

- **Input Embeddings**: The model processes entire sentences as input. Sentences are tokenized and projected into high-dimensional vectors.
- Example: English input "The cat sat on the mat" and "The dog ran fast" tokenized as ["The", "cat", "sat", "on", "the", "mat"] and ["The", "dog", "ran", "fast"].
- Each token is embedded into a vector (e.g., dimension $d = 512$).
- For NLP tasks, how you tokenize is important and can create problems if not done correctly!



Input and Input Embeddings mathematics

- The input sequence of tokens is represented as a matrix $\boldsymbol{X}_{1:t} = [\boldsymbol{X}_1, \ldots, \boldsymbol{X}_t]^\top \in \mathbb{R}^{t \times q}$, where t is the sequence length and q is the input dimension (e.g., one-hot encoded tokens).
- Tokens are embedded in the same way as we embed categorical data (discussed in Lecture 6), i.e.:
- Choose an embedding dimension $b \in \mathbb{N}$, this is a hyper-parameter selected by the modeler, typically $b \ll K$.
- An entity embedding (EE) is defined by $e^{EE} : A \to \mathbb{R}^b, X \mapsto e^{EE}(X)$. In total this entity embedding involves $b \cdot K$ embedding weights.
- Later, we use the notation $\boldsymbol{E}_{1:t}$ for the matrix of embeddings.
- Generally b is much larger than we use for tabular data.

Positional Encoding

- **Positional Encoding**: Fixed sinusoidal vectors are added to input embeddings to encode token positions, preserving sequence order.
- Formula:
  - $\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$
  - $\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$
- Example: For both "The cat sat on the mat" and "The dog ran fast", position 1 ("The" in both) gets the same PE; position 2 ("cat" and "dog") gets the same PE, and so on, differentiating positions regardless of tokens.

Positional Encoding mathematics

- Positional encodings are added to embeddings: $\tilde{\boldsymbol{E}}_{1:t} = \boldsymbol{E}_{1:t} + \boldsymbol{P}_{1:t} \in \mathbb{R}^{t \times d}$, where $\boldsymbol{P}_{1:t} = [\boldsymbol{p}_1, \ldots, \boldsymbol{p}_t]^\top$ and each $\boldsymbol{p}_u \in \mathbb{R}^d$ encodes position u.
- The fixed sinusoidal encoding is defined component-wise for position u and dimension j: $p_{u,j} = \sin(u/10000^{j/d})$ if j even, or $\cos(u/10000^{(j-1)/d})$ if j odd.
- This preserves relative positional information and allows the model to distinguish sequence order without learnable parameters.
- For modern models, we use a more efficient approach called Rotary Position Embeddings (RoPE) which is discussed in Su, Lu, Pan, Wen, and Liu (2021).

Rotary Position Embeddings (RoPE) 1

- **Idea**: Encode *relative* positions by rotating query and key vectors by position-dependent angles, so attention depends on token offsets rather than absolute indices (Su et al., 2021).

- **Math**: Split d into 2D pairs and apply a rotation to each pair. For pair index $i = 1, \ldots, d/2$ use frequency $\omega_i = 10000^{-\frac{2(i-1)}{d}}$ and define the block-diagonal rotation $\boldsymbol{R}(u) = \mathrm{diag}(\boldsymbol{R}_2(u\,\omega_1), \ldots, \boldsymbol{R}_2(u\,\omega_{d/2}))$ with $\boldsymbol{R}_2(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$. Then

$$\tilde{\boldsymbol{q}}_u = \boldsymbol{R}(u)\,\boldsymbol{q}_u, \qquad \tilde{\boldsymbol{k}}_v = \boldsymbol{R}(v)\,\boldsymbol{k}_v.$$

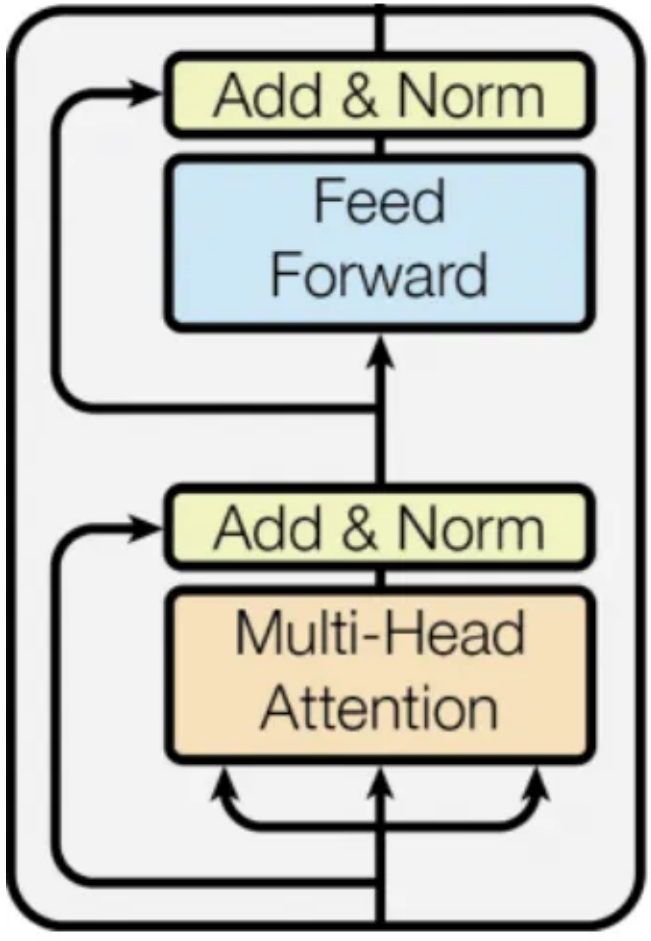
Inner products rotate equivariantly: $\tilde{\boldsymbol{q}}_u^\top \tilde{\boldsymbol{k}}_v = \boldsymbol{q}_u^\top \boldsymbol{R}(u-v)\,\boldsymbol{k}_v$, so attention depends on the *offset* $(u-v)$.

Rotary Position Embeddings (RoPE) 2

- **Intuition**: Rotation encodes phase that advances with position. Dot products compare phases, naturally capturing relative distances; extrapolates to longer sequences since rotations repeat smoothly beyond training range.
- **Benefits**: Better length extrapolation and strong empirical performance with minimal changes to the Transformer.
- **Adoption**: Now standard in modern LLMs (e.g., LLaMA families) (Touvron, Lavril, Izacard, Martinet, et al., 2023; Touvron, Martin, Stone, Albert, et al., 2023).

Encoder Layer

- **Input**: Takes positional encoded embeddings (input embeddings + positional encodings).
- **Multi-Head Attention**: Projects inputs to Queries (Q), Keys (K), and Values (V) via linear layers; applies scaled dot-product attention in parallel heads, then concatenates and projects.
- Residual Connection & Normalization
- Feed-Forward Network
- Residual & Normalization (Again)
- Stacked multiple times (e.g., 6 layers) for deep contextualization.

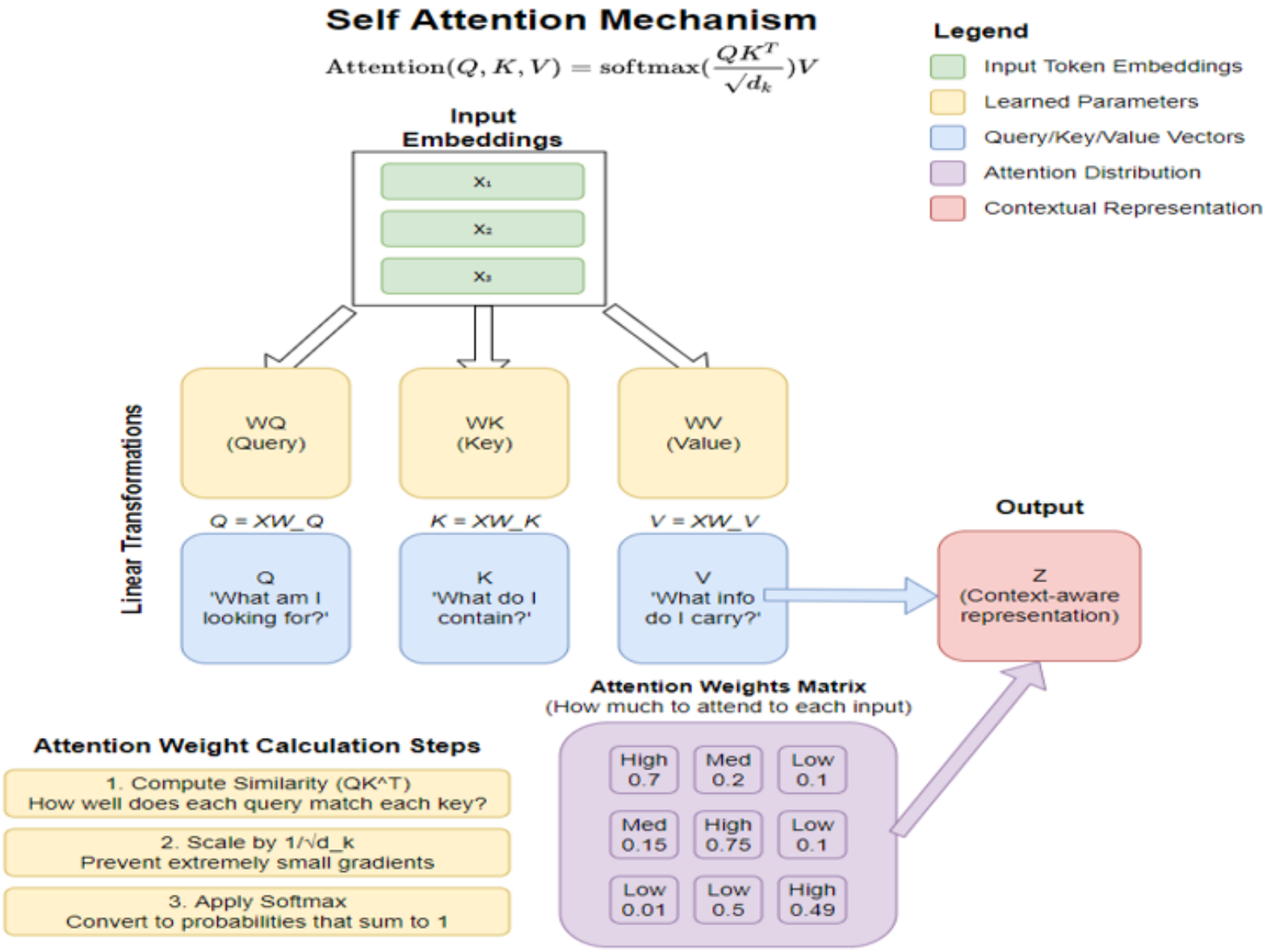


Encoder layer

Self-attention - The Problem

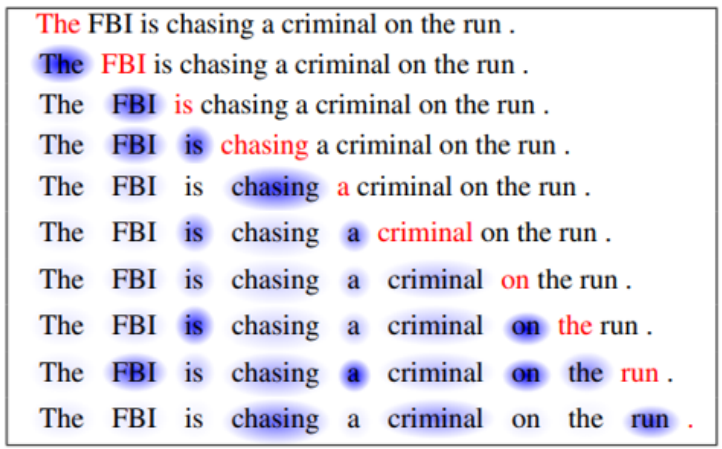
- How can we get the matrix of embeddings to "learn" from its own context?
- Examples:
  - A young driver's car type embedding should have a different value from an older driver
  - A building value embedding should have a different value if it is in a flood zone or if it isn't
- Allow the embeddings to pay attention to their "surroundings"...
- ... looks for matches between queries and keys across different covariates

Self-attention Diagram



Intuitions for self-attention in NLP

- When building models, relationships between covariates and outcomes may depend on context.
- Famous example in non-life insurance – young drivers and male drivers often have increased frequency, but young male drivers may experience even higher frequency => context allow for via interaction effect
- Automated method for building context into models: self-attention i.e. apply attention over inputs; sequence example (Cheng, Dong, & Lapata, 2016)

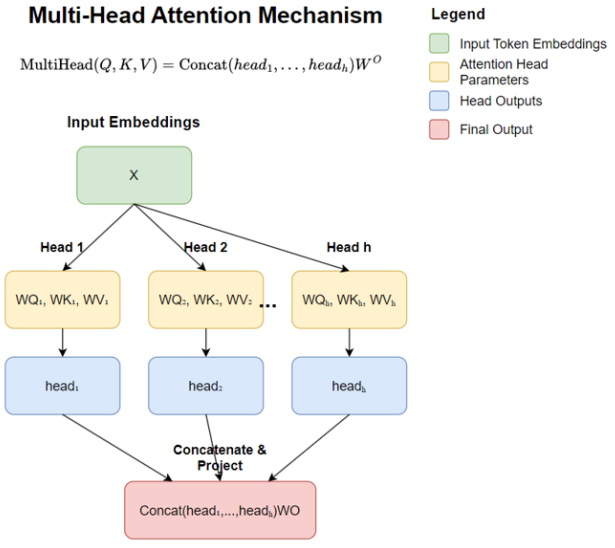


Encoder Layer mathematics

- Input: Positional embeddings $\tilde{E}_{1:t} \in \mathbb{R}^{t \times d}$.
- Multi-head attention:
  - For each head j, compute $\boldsymbol{H}_j = \text{softmax}(\boldsymbol{Q}_j \boldsymbol{K}_j^\top / \sqrt{d}) \boldsymbol{V}_j$
  - Then concatenate and project: $\boldsymbol{H}_{\text{MH}} = \text{Concat}(\boldsymbol{H}_1, \ldots, \boldsymbol{H}_{n_h}) \boldsymbol{W} \in \mathbb{R}^{t \times d}$, where $\boldsymbol{Q}_j, \boldsymbol{K}_j, \boldsymbol{V}_j$ are linear projections of $\tilde{\boldsymbol{E}}_{1:t}$.
- Residual: Add input $\tilde{\boldsymbol{E}}_{1:t} + \boldsymbol{H}_{\text{MH}}$, followed by layer normalization $\boldsymbol{z}^{\text{norm}}(\cdot)$.
- Feed-forward: Time-distributed FNN $\boldsymbol{z}^{t\text{-FNN}}(\cdot) \in \mathbb{R}^{t \times d}$, residual, and normalization.
- Stacked L times for hierarchical representations.

Intuitions for MHA

- Transformers usually apply multiple attention operations
- Each operation is called an "attention head"
- This allows the model to learn different attention operations and value matrices
- Outputs combined at the end before moving further into the model
- Can also stack Transformer layers together to produce a deep model
- Both of these improve model capacity and performance



Layer Normalization = Default for Transformers

- **What**: Normalize each token's hidden features to zero mean and unit variance, then apply learned scale γ and shift β (Ba, Kiros, & Hinton, 2016).
- **Formula**:

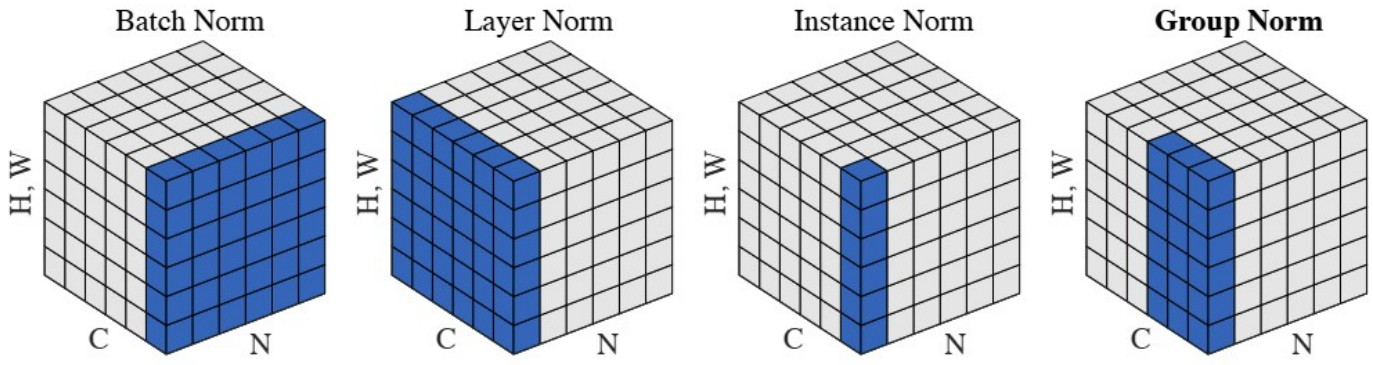
$$\mathrm{LN}(\boldsymbol{h}) = \gamma \odot \frac{\boldsymbol{h} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \quad \mu = \frac{1}{d}\sum_{j=1}^{d} h_j, \quad \sigma^2 = \frac{1}{d}\sum_{j=1}^{d}(h_j - \mu)^2$$

  where statistics are computed *per token across features* (dimension d).

- **Why (for Transformers)**:
  - Stabilizes deep training with residual connections and attention
  - Works with small/variable batch sizes and sequence lengths
  - Same behavior at train and inference (no running averages)

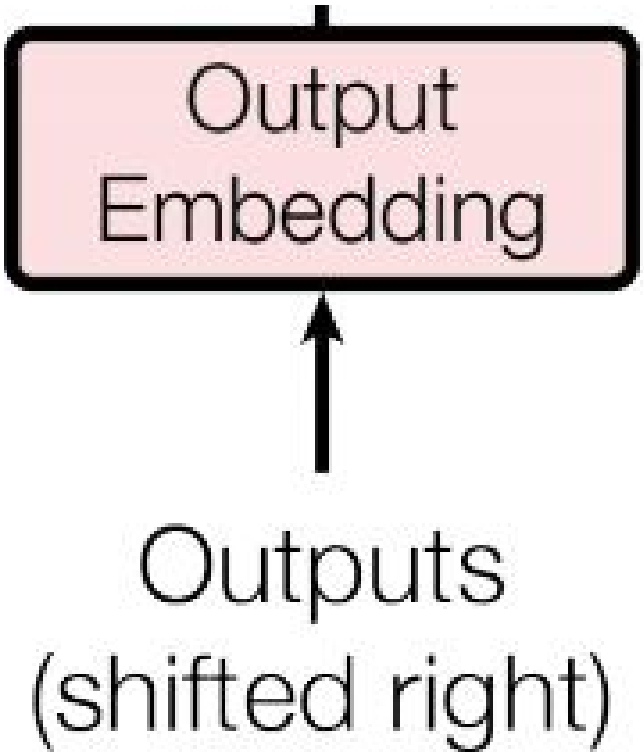
LayerNorm vs BatchNorm

- **How it differs** (see also Wu and He (2018) and image below):
  - **LN**: per-token, across features; independent of batch size; identical at train/inference
  - **BN**: across the batch (and often spatial/time dims); needs large, stable batches; uses running means/vars at inference; brittle for autoregressive decoding (Ioffe & Szegedy, 2015)
- BN is preferred in CNNs where large batches and shared spatial stats are natural



Decoder Input - Output Shifted Right - 1

- **Decoder Input**: During training, the decoder takes the target sequence shifted right (teacher forcing); during inference, it uses previously generated tokens autoregressively.
  - Example: For French target "Le chat s'est assis sur le tapis" (from "The cat sat on the mat"), shifted right as ["<BOS>", "Le", "chat", "s'est", "assis", "sur", "le"].
  - Right shift: prepend "<BOS>" and drop the last token so inputs are ["<BOS>", $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_{s-1}$] and targets are $[\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_s]$.
  - Alignment & no peeking: at position t, input is $\boldsymbol{Y}_{t-1}$ (BOS for t=1) and the label is $\boldsymbol{Y}_t$; a causal mask prevents attending to future tokens.

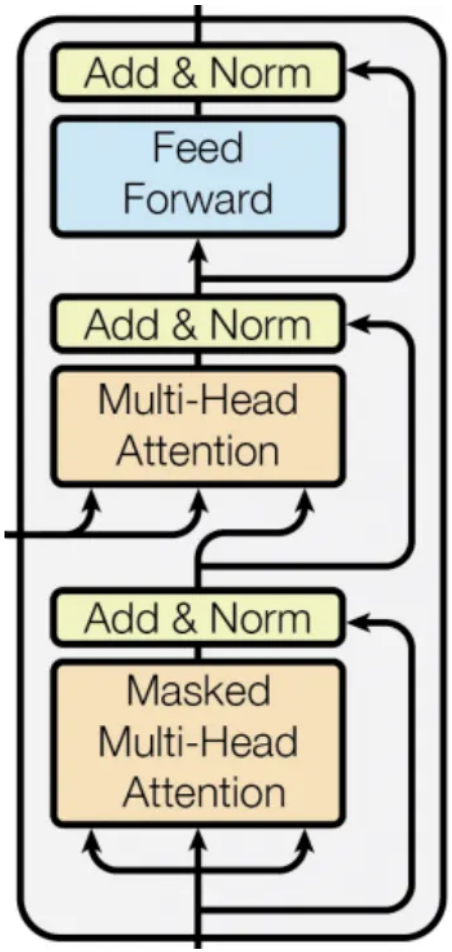


Decoder Input - Output Shifted Right - 2

- **Mathematics**: Target sequence $\boldsymbol{Y}_{1:s} = [\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_s]^\top \in \mathbb{R}^{s \times q}$ is shifted to $\boldsymbol{Y}_{0:s-1} = [\text{BOS}, \boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_{s-1}]^\top \in \mathbb{R}^{s \times q}$; embed to $\boldsymbol{F}_{0:s-1} \in \mathbb{R}^{s \times d}$ and add positions $\tilde{\boldsymbol{F}}_{0:s-1} = \boldsymbol{F}_{0:s-1} + \boldsymbol{P}_{0:s-1}$. Objective: maximize $\sum_{t=1}^{s} \log p(\boldsymbol{Y}_t \mid \boldsymbol{Y}_{0:t-1})$.

Decoder Layer: Masked Self-Attention

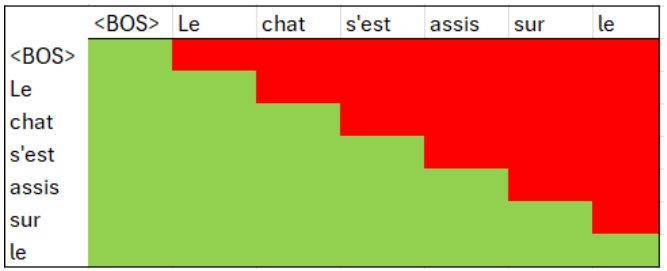
- **Input**: Takes positionally encoded decoder embeddings $\tilde{\boldsymbol{F}}_{0:s-1} = \boldsymbol{F}_{0:s-1} + \boldsymbol{P}_{0:s-1}$.
- **Masked Multi-Head Attention**: Self-attention with causal mask; project to $\boldsymbol{Q}$, $\boldsymbol{K}$, $\boldsymbol{V}$ from decoder input and apply scaled dot-product attention.
  - Example: For shifted "Le chat s'est assis sur le tapis", mask ensures "Le" attends only to "<BOS>", "chat" to "<BOS>" and "Le", etc.
- **Mathematics**: With $\tilde{\boldsymbol{F}}_{0:s-1} \in \mathbb{R}^{s \times d}$, compute $\boldsymbol{A}' = \boldsymbol{Q}\boldsymbol{K}^\top / \sqrt{d} + \boldsymbol{M}$ where $\boldsymbol{M}$ is lower-triangular with $-\infty$ above diagonal; $\boldsymbol{A} = \text{softmax}(\boldsymbol{A}')$, output $\boldsymbol{H} = \boldsymbol{A}\boldsymbol{V}$. Residual $\tilde{\boldsymbol{F}}_{0:s-1} + \boldsymbol{H}$, then layer norm $\boldsymbol{z}^{\text{LN}}(\cdot)$.

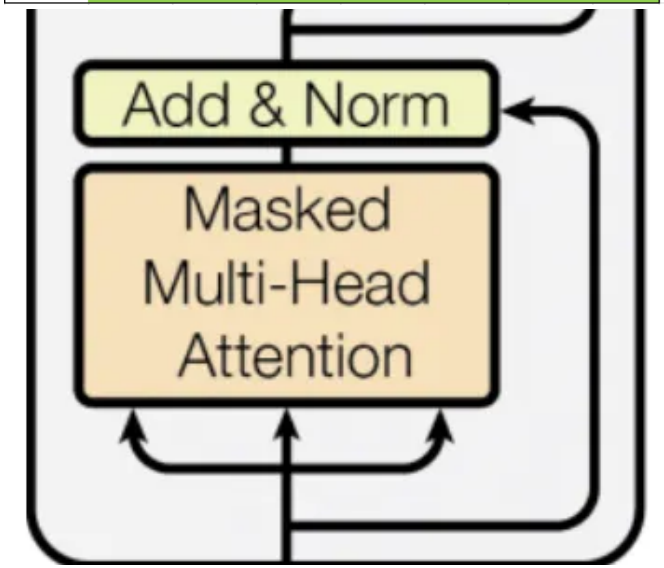


Decoder layer

Decoder Layer: Masking in Masked Multi-Head Attention - 1

- **Masking Mechanism**: Enforces causality in decoder self-attention; prevents attending to future positions.
  - After computing $\boldsymbol{A}' = \boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d}$, add mask $\boldsymbol{M}$: $m_{u,v} = 0$ for $v \leq u$, and $m_{u,v} = -\infty$ otherwise.
  - Apply softmax row-wise to $\boldsymbol{A}' + \boldsymbol{M}$; masked positions receive (near) zero weight.
  - Shape: $\boldsymbol{M} \in \mathbb{R}^{s \times s}$ lower triangular (including diagonal).
- **Example**: For shifted input ["<BOS>", "Le", "chat", "s'est", "assis", "sur", "le"] (from "The cat sat on the mat" translation), position 3 ("s'est") attends only to positions 1-3; positions 4-7 are masked (red in image).



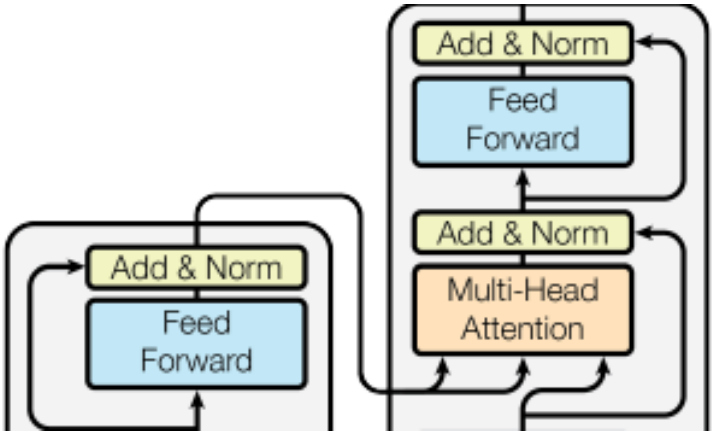


Decoder Layer: Masking in Masked Multi-Head Attention - 2

- Masking: For attention matrix $\boldsymbol{A}' = \boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d} \in \mathbb{R}^{s\times s}$, add mask $\boldsymbol{M} \in \mathbb{R}^{s\times s}$ where $m_{u,v} = 0$ if $v \leq u$, else $m_{u,v} = -\infty$.
- Softmax: $a_{u,v} = \exp(a'_{u,v} + m_{u,v})/\sum_{k=1}^{s} \exp(a'_{u,k} + m_{u,k}) \in (0, 1)$, ensuring row sums to 1 and no future attention.
- Output: $\boldsymbol{H} = \boldsymbol{A}\boldsymbol{V} \in \mathbb{R}^{s\times d}$.
- Residual Connection & **Layer Normalization (LN)**

Decoder Layer: Cross-Attention and FFN - 1

- **Cross Multi-Head Attention**: Projects decoder output to Q; uses Keys (K) and Values (V) from encoder's high-dimensional representations; enables conditioning on source.
  - Example: Decoder Q from "Le chat..." attends to encoder K/V from "The cat sat on the mat" for translation alignment.
- Residual Connection & Normalization
- Feed-Forward Network
- Residual & Normalization (Again)
- Stacked multiple times (e.g., 6 layers) for autoregressive generation.



Decoder Layer: Cross-Attention and FFN - 2

- Cross-attention: $\boldsymbol{Q}$ from decoder state, $\boldsymbol{K}, \boldsymbol{V}$ from encoder output $\boldsymbol{O}_{1:t} \in \mathbb{R}^{t \times d}$; compute $\boldsymbol{H} = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top / \sqrt{d})\boldsymbol{V} \in \mathbb{R}^{s \times d}$.
- Residual and normalization: Add previous state and apply $\boldsymbol{z}^{\text{norm}}(\cdot)$.
- Feed-forward: Time-distributed FNN $\boldsymbol{z}^{s\text{-FNN}}(\cdot) \in \mathbb{R}^{s \times d}$, residual, and normalization.
- Stacked L times.

Final Prediction and Loss Calculation - 1

- **Decoder Output**: After stacked decoder layers, obtain contextualized vectors (dimension d) for each position in the sequence.
- **Linear Projection**: Apply a linear layer to project these vectors to the vocabulary size, producing logits for each token position.
- **Softmax**: Convert logits to probability distributions over the vocabulary for each position.
- **Prediction and Comparison**: During training, predictions are compared to the target sequence (original, not shifted) using cross-entropy loss; argmax or sampling for generation.
    - Example: For shifted input ["<BOS>", "Le", "chat", "s'est", "assis", "sur", "le"], model predicts logits/softmax for next tokens: after "<BOS>" predict "Le", after "Le" predict "chat", etc.; compared against target ["Le", "chat", "s'est", "assis", "sur", "le", "tapis"].

Final Prediction and Loss Calculation - 2

- Decoder output: Contextualized matrix $\boldsymbol{G}_{0:s-1} \in \mathbb{R}^{s \times d}$.
- Linear projection: Time-distributed layer $\boldsymbol{L}_{1:s} = \boldsymbol{z}^{s\text{-FNN}}(\boldsymbol{G}_{0:s-1}) \in \mathbb{R}^{s \times v}$, where v is vocabulary size, producing logits.
- Softmax: Probabilities $P_{u,j} := \exp(l_{u,j}) / \sum_{k=1}^{v} \exp(l_{u,k})$ for position u and token j.
- Loss: Cross-entropy between predictions and true targets $\boldsymbol{Y}_{1:s}$, e.g., $-\sum_{u=1}^{s} \sum_{j=1}^{v} y_{u,j} \log p_{u,j}$.

1 Introduction to Transformers

2 Understanding Attention is All You Need

3 Seq2Seq to Tabular Data

4 Credibility

5 Example + Interpretability

6 Improvements

7 Conclusions

From Sequence-to-Sequence to Tabular Transformers - 1

- The Transformer in *Attention Is All You Need* applies self-attention to sequential text data.
- This enables dynamic interactions between positions, allowing the model to focus on relevant parts of the sequence adaptively.
- Can we extend this architecture to tabular data, treating features as "sequences"?
- Key adaptations from *Attention Is All You Need*include:
  - Positional encoding
  - Independent processing of input "sequences"
  - Encoder Structure and Multihead Attention

From Sequence-to-Sequence to Tabular Transformers - 2

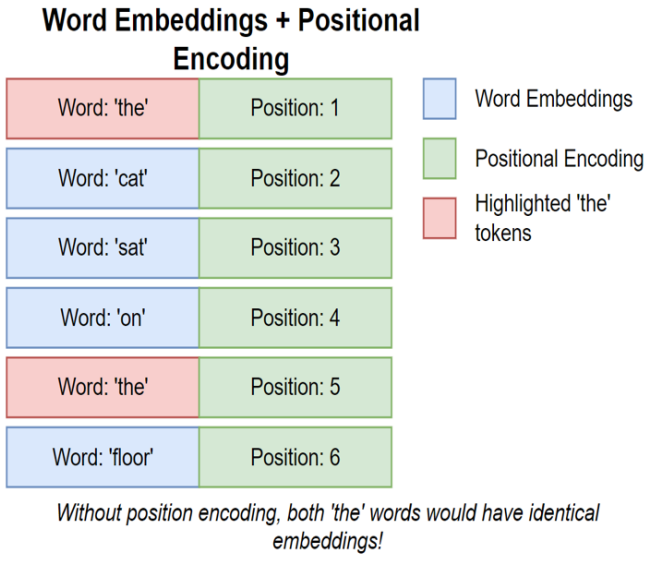
- In sequence models, input is a tensor $\boldsymbol{X}_{1:t} \in \mathbb{R}^{t \times q}$, where t is sequence length and q is input dimension; self-attention computes interactions via $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ projections.
- For tabular data with f features, treat as "sequence" $\boldsymbol{X}_{1:f} \in \mathbb{R}^{f \times q}$, enabling attention over features instead of time steps.
- Adaptations preserve core mappings: Attention head $H(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top / \sqrt{q})\boldsymbol{V} \in \mathbb{R}^{f \times q}$, but now capturing feature interactions.

Key References: Tabular Transformers

- TabTransformer for contextual embeddings of categorical features Huang, Khetan, Cvitkovic, and Karnin (2020).
- FT-Transformer (feature tokenization + Transformer) Gorishniy, Rubachev, Khrulkov, and Babenko (2021).
- TabNet: attentive, interpretable tabular learning Arik and Pfister (2021).
- AutoInt: self-attentive feature interaction learning Song et al. (2019).
- TransTab: transfer across heterogeneous tables Wang and Sun (2022).

Self attention and ordering

- Tabular embeddings have no natural ordering...
- ... so we cannot apply the sinusoidal positional encodings to the embeddings
- Solution: add an extra set of positional embeddings (encodings) which indicate from which column of the tabular data the embedding arises from
- Done here by simply learning a new embedding for each column



From Positional to Feature Encoding mathematics

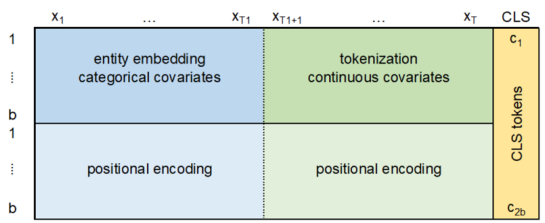
- In sequences, positional encodings $\boldsymbol{P}_{1:t} = [\boldsymbol{p}_1, \ldots, \boldsymbol{p}_t]^\top \in \mathbb{R}^{t \times d}$ are added to embeddings $\boldsymbol{E}_{1:t}$, with fixed sinusoidal $p_{u,j} = \sin(u/10000^{j/d})$ or cosine.

- For tabular data with f features, use learnable feature encodings $\boldsymbol{P}_{1:f} \in \mathbb{R}^{f \times d}$, added to feature embeddings $\boldsymbol{E}_{1:f}$, yielding $\tilde{\boldsymbol{E}}_{1:f} = \boldsymbol{E}_{1:f} + \boldsymbol{P}_{1:f}$.

- Each p_u is trained, encoding feature identity (e.g., 'age' gets fixed p_u across batch), enabling attention to distinguish features without inherent order.

A special type of embedding – the CLS token

- Output of self-attention is a matrix of contextualized embeddings
- This could be used directly for prediction BUT this might (often!) overfit the data due to the flexibility allowed through the Transformer model
- Can we condense the information into a new vector we can use to make predictions?
- This idea is from the BERT paper (Devlin et al., 2019), which adds a **CLS token** to the embeddings
- This gets updated with information from all the other embeddings, but has a much lower dimension => Will make more robust out of sample predictions
- **Usually initialized at random values**

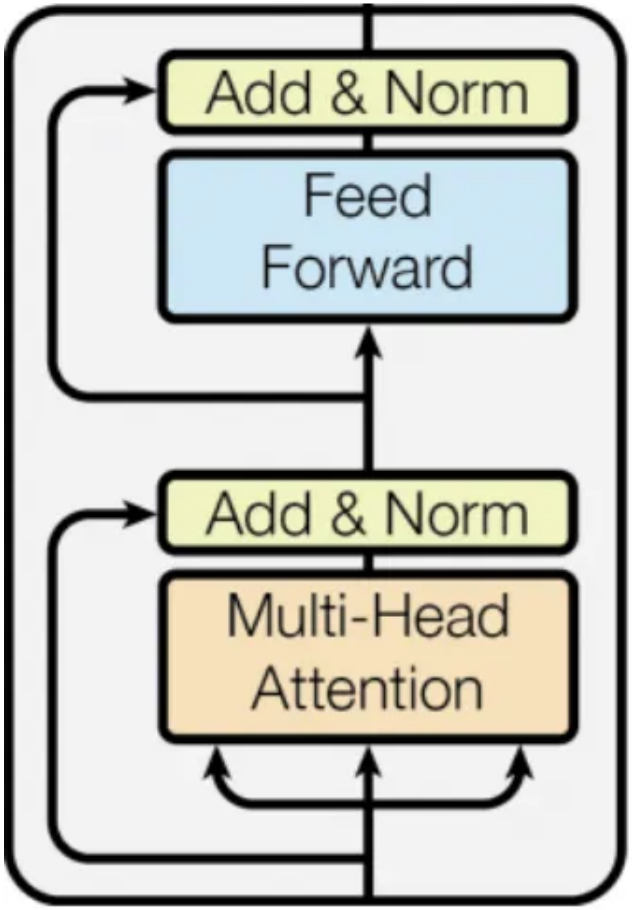


CLS Token mathematics

- CLS token: Learnable vector $\boldsymbol{c} \in \mathbb{R}^d$, concatenated to feature embeddings $\tilde{\boldsymbol{E}}_{1:f} \in \mathbb{R}^{f \times d}$, yielding augmented input $\tilde{\boldsymbol{E}}_{0:f} = [\boldsymbol{c}, \tilde{\boldsymbol{E}}_1, \ldots, \tilde{\boldsymbol{E}}_f]^\top \in \mathbb{R}^{(f+1) \times d}$.
- After Transformer layers, extract processed CLS $\boldsymbol{h}_0 \in \mathbb{R}^d$ from output $\boldsymbol{H}_{0:f} \in \mathbb{R}^{(f+1) \times d}$.
- Prediction: Feed-forward $\boldsymbol{z}^{\text{FNN}}(\boldsymbol{h}_0) \in \mathbb{R}^p$, where p is output dimension (e.g., scalar for regression).

Encoder Structure for Tabular Data

- The encoder layer from the sequence-to-sequence Transformer is sufficient for handling tabular data.
- For tabular tasks, a much simpler "decoder" suffices compared to seq2seq models.
- It extracts the CLS token (or vector) from the final Transformer (encoder) layer and passes it through a feed-forward neural network to generate predictions.



Encoder layer

Encoder Structure for Tabular Data mathematics

- Input: Augmented embeddings $\tilde{\boldsymbol{E}}_{0:f} \in \mathbb{R}^{(f+1)\times d}$ (including CLS).
- Multi-head attention: Compute $\boldsymbol{H}_{\text{MH}} = \text{Concat}(\boldsymbol{H}_1, \ldots, \boldsymbol{H}_{n_h})\boldsymbol{W} \in \mathbb{R}^{(f+1)\times d}$, with each $\boldsymbol{H}_j = \text{softmax}(\boldsymbol{Q}_j \boldsymbol{K}_j^\top / \sqrt{d})\boldsymbol{V}_j$; add residual and normalize $\boldsymbol{z}^{\text{norm}}(\tilde{\boldsymbol{E}}_{0:f} + \boldsymbol{H}_{\text{MH}})$.
- Feed-forward: Time-distributed $\boldsymbol{z}^{(f+1)\text{-FNN}}(\cdot) \in \mathbb{R}^{(f+1)\times d}$, residual, and normalization; stack L layers.
- Output: Extract CLS for final FNN prediction.

Self-Attention for Tabular Data mathematics

- From augmented input $\tilde{\boldsymbol{E}}_{0:f} \in \mathbb{R}^{(f+1)\times d}$, project to $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{(f+1)\times d}$ via time-distributed FNNs: $\boldsymbol{Q} = \boldsymbol{z}^{(f+1)\text{-FNN}_Q}(\tilde{\boldsymbol{E}}_{0:f})$, similarly for $\boldsymbol{K}, \boldsymbol{V}$.

- Attention: $\boldsymbol{A} = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d}) \in \mathbb{R}^{(f+1)\times(f+1)}$, with elements $a_{u,s} = \exp(\boldsymbol{q}_u^\top \boldsymbol{k}_s/\sqrt{d}) / \sum_k \exp(\boldsymbol{q}_u^\top \boldsymbol{k}_k/\sqrt{d}) \in (0,1)$.

- Output: $\boldsymbol{H} = \boldsymbol{A}\boldsymbol{V} \in \mathbb{R}^{(f+1)\times d}$, where each row is a weighted average of values, capturing feature (and CLS) interactions.

1 Introduction to Transformers

2 Understanding Attention is All You Need

3 Seq2Seq to Tabular Data

4 Credibility

5 Example + Interpretability

6 Improvements

7 Conclusions

Credibility

- Credibility = classical approach to weighting between:
  - Overall population experience
  - Experience of particular policyholder

$$p_n = (1 - \alpha)\, p_{n-1} + \alpha \cdot r_{n-1}$$

p_k = rate for period k
r_k = loss ratio for period k
α is called *credibility* and assumed to be a function of the volume V, mostly

$$\alpha(V) = \frac{V}{V+k} \quad \text{or} \quad \alpha(V) = \begin{cases} 1 & \text{if } V \geqslant V_0 \\ \sqrt{\frac{V}{V_0}} & \text{if } V < V_0 \end{cases}$$

V_0 is called the *volume of full credibility*.

- Celebrated Bühlmann credibility model provides a distribution free best linear approximation to this pricing problem!
- *Credibility also means more generally the amount of weight we give to a particular piece of data*

INTRODUCTORY REPORT

EXPERIENCE RATING AND CREDIBILITY

HANS BÜHLMANN
Zürich

$$(1 - b) \cdot E[\mu(\vartheta)] + b \cdot \overline{X}$$

$$b = \frac{n}{n+k}$$

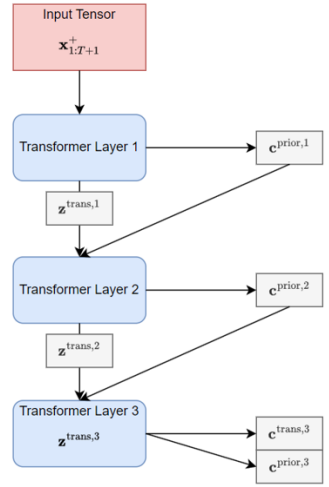
$$k = \frac{E[\sigma^2(\vartheta)]}{\mathrm{Var}[\mu(\vartheta)]}$$

Inducing Credibility into the Transformer

- **Main idea is combine the prior CLS parameters and the data guided information into a credibility formula for optimally training the model.**
- CLS token is usually randomly initialized => Transformer can learn whatever value makes the model fit the data closely
- Can we calibrate the CLS token to produce the portfolio mean = play the role of prior information?
- Create two paths in the Transformer model:
  - Path 1 – predict using the CLS token directly = portfolio mean
  - Path 2 – predict using the CLS token after updates with covariate information
  - Choose between Path 1 and Path 2 using a random factor
- We want the model to learn good covariate information, so choose path 2 more often

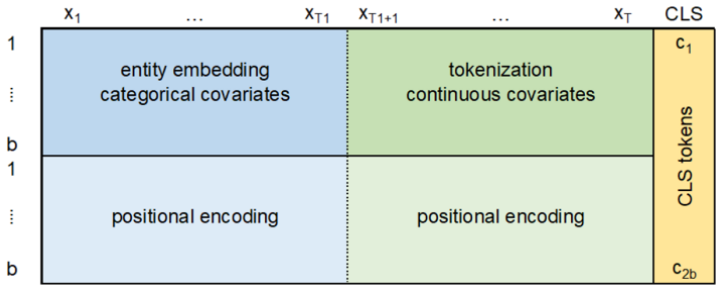
$$\boldsymbol{c}^{\text{cred}} = Z\,\boldsymbol{c}^{\text{trans}} + (1-Z)\,\boldsymbol{c}^{\text{prior}} \in \mathbb{R}^{2b}$$

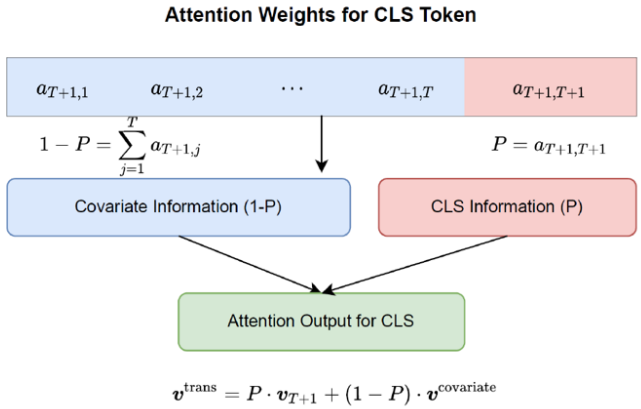
$$Z \sim \text{Bernoulli}(\alpha)$$



Self attention then becomes a credibility formula!

- We have now calibrated the CLS token to be the portfolio mean
- In the attention mechanism, we will give a probability weight P to this value of the CLS token...
- ... and a weight of (1-P) to the embedded covariate information
- => self-attention now performs a credibility operation between the prior information and covariates!
- Instead of using the usual Bühlmann formula, we rely on the self-attention mechanism to weight appropriately between:
  - Prior information = portfolio average
  - Covariate information = from embeddings





Summary of changes w.r.t Vanilla Transformer

- Added a CLS token that we will use for prediction
- **Credibility during training:** When **training** the model, use a credibility principle:
  - Select the CLS token with probability α
  - Select the transformed CLS token with probability (1-α)
- Use whichever CLS token is selected to make predictions
- **Latent credibility:** this recalibrates the model such that self attention **becomes a credibility formula**

1 Introduction to Transformers

2 Understanding Attention is All You Need

3 Seq2Seq to Tabular Data

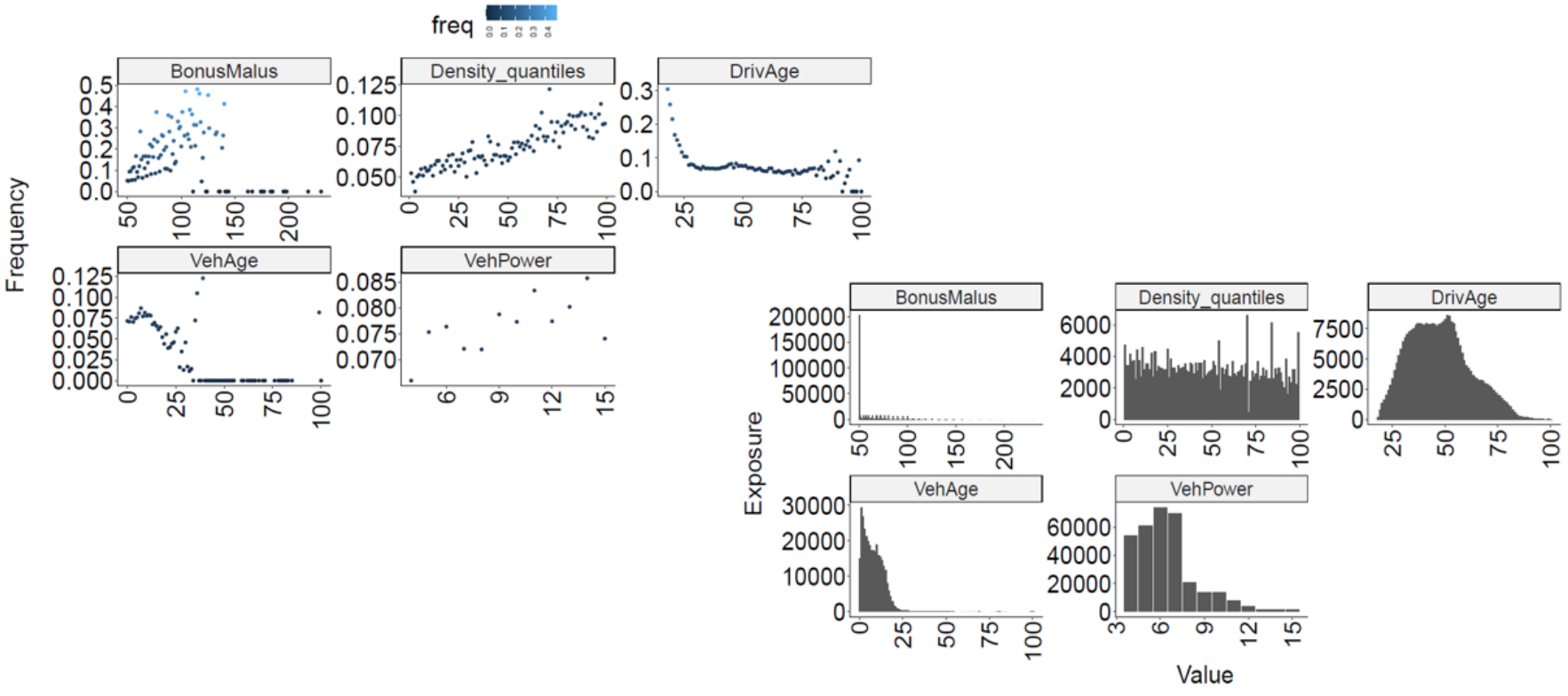
4 Credibility

5 Example + Interpretability

6 Improvements

7 Conclusions

French MTPL data - Detail



Models

- French MTPL data has 5 continuous and 4 categorical covariates
- All 9 covariates embedded using approach described above
- Predicting claims counts, so used Poisson loss
- Used credibility probability = 90% during training
- How complex and big does a Transformer need to be?
- Here, we use a total of 1 746 parameters and 1 073 of these are in the Transformer model => **can build small versions of these models**
- Results improve on GLM/FNN/other attempt to build a Transformer model in Brauer (2024)

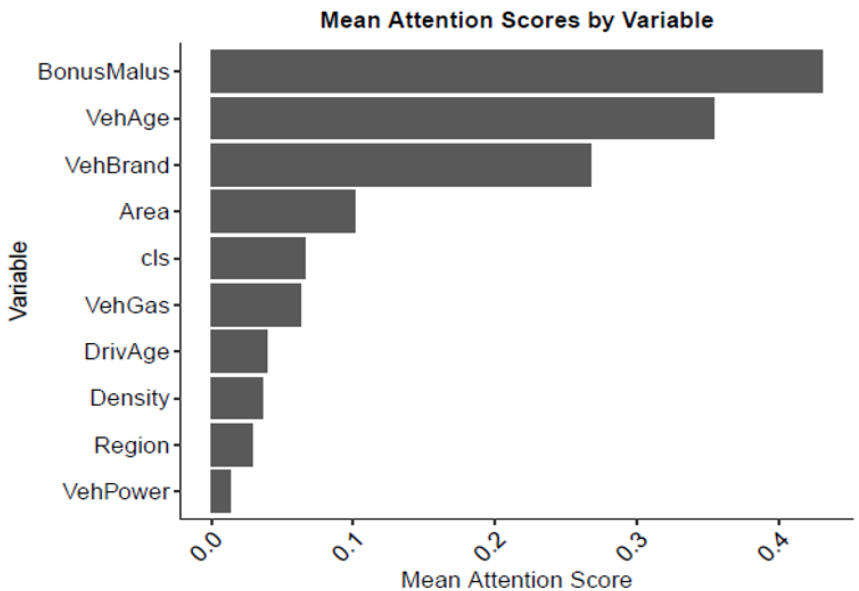
Set	N	Exposure	Claims	Frequency
learn	610,206	322,392	23,737	0.0736
test	67,801	35,967	2,644	0.0735

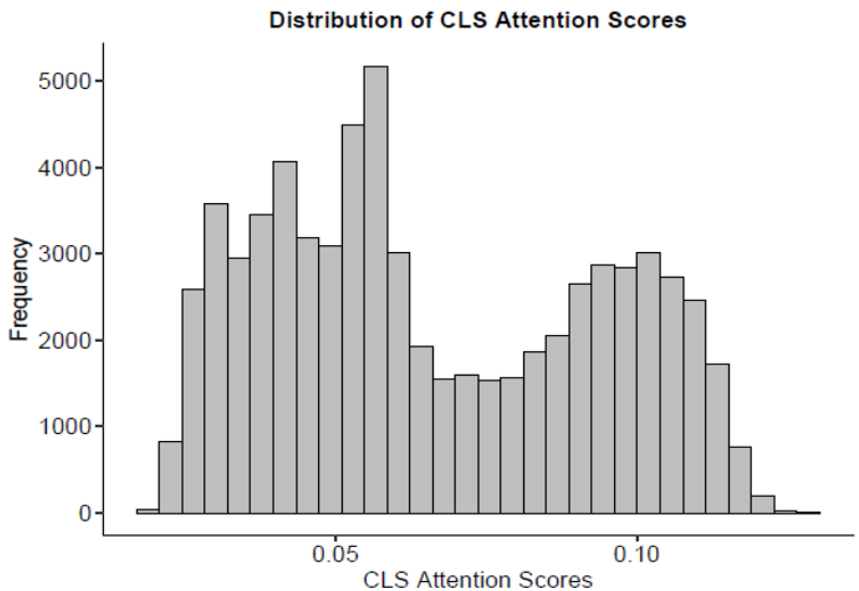
Module	Variable/layer	# Weights
Feature tokenizer (raw input tensor)	$x^{\circ}_{1:9}$	405
Positional encoding	$e^{\text{pos}}_{1:9}$	45
CLS tokens	c	10
Time-distributed normalization layer	z^{norm}	20
Credibility Transformer	c^{cred}	1,073
FNN decoder	$z^{(2:1)}$	193

Architecture	# Param.	Out-of-sample Poisson loss
Poisson null (no covariates)	1	25.445
Poisson GLM3	50	24.102
Ensemble plain-vanilla FNN	792	23.783
Ensemble Feature Tokenizer Transformer (FTT)	27,133	23.759
Ensemble Credibility Transformer: `nadam`	1,746	23.717
Ensemble Credibility Transformer: `NormFormer`	1,746	23.711

Interpreting the model

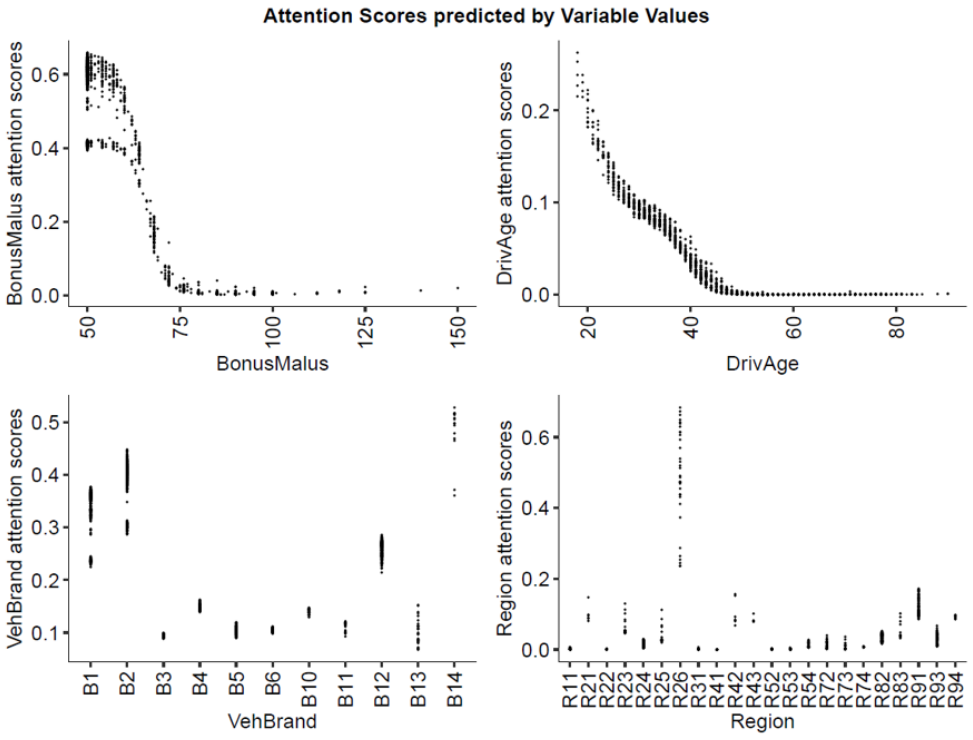
- Rich information available directly from the Credibility Transformer on how decisions have been made
- How much attention been given to the various covariates and extent to which we focus on prior information
- Scores align to what we would expect on personal lines

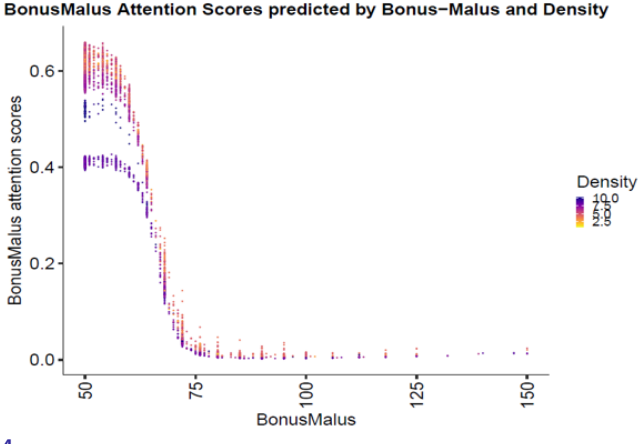




Attention depends on the covariate values

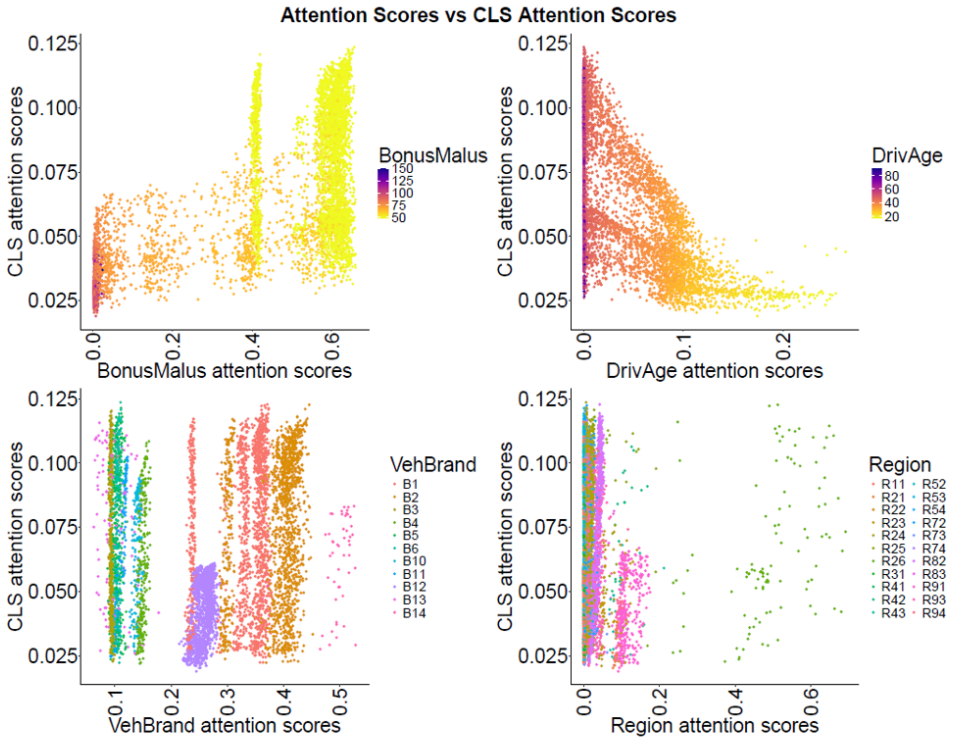
- The attention coefficients are strongly related to the values the covariate takes...
- ... i.e. the model automatically assigns extra weight to variables depending on the information they convey
- Attention also varies by value of other covariates





Prior weights also depend on the covariates

- Strong relationship between the covariate values and the CLS attention scores
- I.e. for how the credibility given to the portfolio experience varies with the values of the other covariates
- Highest credibility is given to the portfolio experience when:
  - BonusMalus scores are low
  - DrivAge is middle-aged
  - VehBrand is not B12
  - In several of the Regions in the dataset



1 Introduction to Transformers

2 Understanding Attention is All You Need

3 Seq2Seq to Tabular Data

4 Credibility

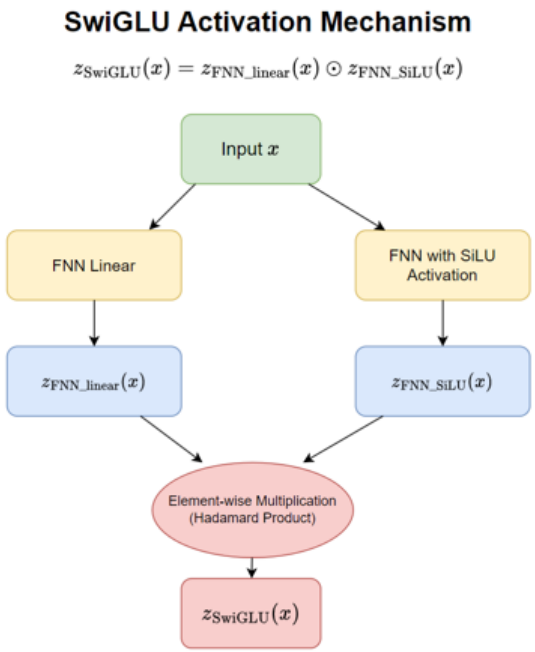
5 Example + Interpretability

6 Improvements

7 Conclusions

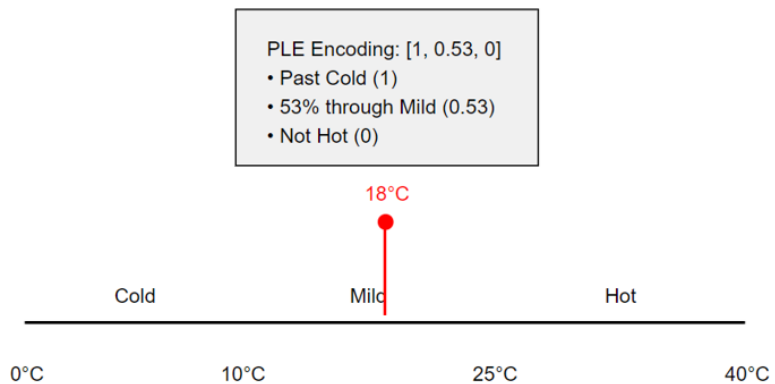
Attention all the way down

- Traditional neural networks treat all inputs as equally important
- In reality, some features matter more than others
- We need yet ANOTHER way to selectively focus on what's important
- Gated Linear Units (GLU)
  - Acts like a smart filter for network calculations
  - Can learn which inputs are important and which aren't
  - Automatically down-weights or removes less relevant information
- Uses two parallel networks:
  - One processes the data normally
  - Another decides how important each piece is (creates weights between 0 and 1)
  - Multiplies these together to get final output



The Challenge with Continuous Variables

- Need smarter way to handle continuous numbers like age or income
- Want to preserve the natural ordering of numbers
- Need to capture both local and global patterns in the data
- Piecewise Linear Encoding (PLE) of Gorishny et al.
  - Divides the range of values into "bins" (like sorting items into boxes)
  - If value greater than bin boundary => set all values to 1
  - Each value knows which bin it belongs to AND where it sits within that bin
  - Like having both a postal code (bin) and street address (position in bin)
  - Creates a smarter version of one-hot encoding for continuous values



Results

- Performance of the Credibility Transformer significantly enhanced by making these changes
- Despite using a "state of the art" setup, credibility still improves over the vanilla Transformer results ($\alpha = 100\%$)!

Model	In-sample Poisson loss	Out-of-sample Poisson loss
Ensemble Poisson plain-vanilla FNN	23.691	23.783
Ensemble Credibility Transformer (best-performing)	23.562	23.711
Improved Credibility Transformer with $\alpha = 90\%$	23.533 ($\pm$ 0.058)	23.670 ($\pm$ 0.031)
Ensemble Credibility Transformer ($\alpha = 90\%$)	23.454	23.587
Improved Credibility Transformer with $\alpha = 95\%$	23.557 ($\pm$ 0.058)	23.676 ($\pm$ 0.027)
Ensemble Credibility Transformer ($\alpha = 95\%$)	23.465	23.593
Improved Credibility Transformer with $\alpha = 98\%$	23.544 ($\pm$ 0.042)	23.670 ($\pm$ 0.032)
Ensemble Credibility Transformer ($\alpha = 98\%$)	23.460	23.577
Improved Credibility Transformer with $\alpha = 100\%$	23.535 ($\pm$ 0.051)	23.689 ($\pm$ 0.044)
Ensemble Credibility Transformer ($\alpha = 100\%$)	23.447	23.607

1 Introduction to Transformers

2 Understanding Attention is All You Need

3 Seq2Seq to Tabular Data

4 Credibility

5 Example + Interpretability

6 Improvements

7 Conclusions

Summary

- Introduced the Transformer architecture...
- ...which we enhanced with dual credibility mechanisms:
  - Explicit mechanism through probabilistic CLS token selection during training (α parameter)
  - Implicit mechanism through attention weights functioning as learned credibility formula
- Demonstrated integration of modern deep learning components:
  - Multi-head attention
  - Deep architecture with multiple Transformer layers
  - Gated Linear Units (GLU) for feature importance
  - Differentiable Piecewise Linear Encoding (PLE) for continuous covariates

Conclusions

- What lessons can we draw from this work?
- Even in a state of the art deep learning model, applying classical actuarial principles can lead to gains in model performance
- Actuarial science and modern machine learning aren't separate disciplines...
- ... but rather, classical actuarial principles like credibility still have deep relevance in the age of deep learning.

References I

Arik, S. Ö., & Pfister, T. (2021). Tabnet: Attentive interpretable tabular learning. In *Proceedings of the aaai conference on artificial intelligence.* Retrieved from https://arxiv.org/abs/1908.07442

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*. Retrieved from https://arxiv.org/abs/1607.06450

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of the international conference on learning representations (iclr).* Retrieved from https://arxiv.org/abs/1409.0473 (arXiv:1409.0473)

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacl-hlt.* Retrieved from https://arxiv.org/abs/1810.04805

References II

Geva, M., et al. (2021). Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2110.02834*. Retrieved from https://arxiv.org/abs/2110.02834

Gorishniy, Y., Rubachev, I., Khrulkov, V., & Babenko, A. (2021). Revisiting deep learning models for tabular data. *arXiv preprint arXiv:2106.11959*. Retrieved from https://arxiv.org/abs/2106.11959

Huang, X., Khetan, A., Cvitkovic, M., & Karnin, Z. (2020). Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*. Retrieved from https://arxiv.org/abs/2012.06678

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd international conference on machine learning*. Retrieved from https://arxiv.org/abs/1502.03167

References III

Song, W., et al. (2019). Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th acm international conference on information and knowledge management.* Retrieved from https://arxiv.org/abs/1810.11921 doi: 10.1145/3357384.3357925

Su, J., Lu, Y., Pan, S., Wen, B., & Liu, Y. (2021). Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*. Retrieved from https://arxiv.org/abs/2104.09864

Touvron, H., Lavril, T., Izacard, G., Martinet, X., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*. Retrieved from https://arxiv.org/abs/2302.13971

Touvron, H., Martin, L., Stone, K., Albert, P., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*. Retrieved from https://arxiv.org/abs/2307.09288

References IV

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems.* Retrieved from https://arxiv.org/abs/1706.03762

Wang, Z., & Sun, J. (2022). Transtab: Learning transferable tabular transformers across tables. *arXiv preprint arXiv:2205.09328*. Retrieved from https://arxiv.org/abs/2205.09328

Wu, Y., & He, K. (2018). Group normalization. In *Proceedings of the european conference on computer vision (eccv).* Retrieved from https://arxiv.org/abs/1803.08494